Docket No.: POU920030155US1

Inventor: Bruce M. Potter

Title: METHOD, SYSTEM AND PROGRAM PRODUCTS FOR UPDATING EVENT DATA ON A PAGE USING CODE UNDERSTOOD NATIVELY BY A BROWSER

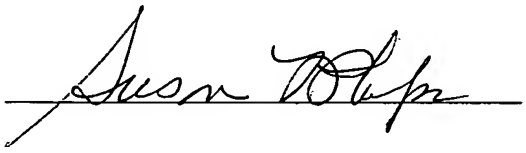
APPLICATION FOR UNITED STATES

LETTERS PATENT

"Express Mail" Mailing Label No.: ER 363668987 US
Date of Deposit: December 18, 2003

I hereby certify that this paper is being deposited with the United States Postal Service as "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to: Mail Stop PATENT APPLICATION, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Name: SUSAN L. PHELPS

Signature: 

INTERNATIONAL BUSINESS MACHINES CORPORATION

METHOD, SYSTEM AND PROGRAM PRODUCTS FOR UPDATING EVENT DATA ON A PAGE USING CODE UNDERSTOOD NATIVELY BY A BROWSER

Technical Field

[0001] This invention relates in general to updating event data, and more particularly, to updating event data on a page of a computing environment.

Background of the Invention

[0002] Many applications have adopted a web browser-based interface employed by a user to interact with the application. These interfaces typically display the application data and provide buttons and links to instruct the application to perform operations. Often, it is also desirable for this web interface to dynamically update the application information that is being displayed, as soon as it changes in the application. This is sometimes called monitoring information, or event driven display. However, this is difficult to accomplish with the web interface, because it is primarily a "pull" model in which new pages are displayed only when the user requests them. There are several ways that web interface developers have attempted to overcome this restriction, but all of the approaches have drawbacks. Due to the inherent limitations of the HyperText Transfer Protocol (HTTP) model, it is not simple to overcome this problem.

[0003] One common solution to this problem is to use a META refresh tag to refresh the entire application page periodically. This approach, however, uses a large amount of network bandwidth and puts a sizable load on the server because the whole page must be generated and downloaded each time it is refreshed. Refreshing an entire page causes another problem if the page has any user input fields. If the page starts to reload when the user has typed in some input but has not yet submitted it, the input will be lost. This is because the input fields are reloaded in the browser and the fields have no knowledge of the data that was just typed.

[0004] Another common solution is to write a Java programming language applet that will display the application data in the page. This has two drawbacks. First, it usually takes a long time for the applet code to download and start up because all the Java base classes need to be downloaded as well. Second, the applet needs to open a network connection to the server in order to allow the server to "push" down data when events occur. This is undesirable because security authentication must be programmed into the newly opened connection and because the port number used by this connection may not be allowed through firewalls existing between the server and the browser.

[0005] Based on the foregoing, a need still exists for an enhanced technique to dynamically update application information that is being displayed by a browser.

Summary of the Invention

[0006] The shortcomings of the prior art are overcome and additional advantages are provided through the provision of a method of updating event data on a page of a computing environment. The method includes, for instance, periodically retrieving event data using a refresh frame of a page; and updating a portion of a data frame of the page with the event data using code understood natively by a browser, wherein the browser displays the page.

[0007] Systems and computer program products corresponding to the above-summarized method are also described and claimed herein.

[0008] Additional features and advantages are realized through the techniques of the present invention. Other embodiments and aspects of the invention are described in detail herein and are considered a part of the claimed invention.

Brief Description of the Drawings

[0009] The subject matter which is regarded as the invention is particularly pointed out and distinctly claimed in the claims at the conclusion of the specification. The foregoing and other objects, features, and advantages of the invention are apparent from the following detailed description taken in conjunction with the accompanying drawings in which:

[0010] FIG. 1 depicts one embodiment of a computing environment incorporating and using one or more aspects of the present invention;

[0011] FIG. 2 depicts one embodiment of logic associated with updating event data on a single page of a computing environment, in accordance with an aspect of the present invention; and

[0012] FIG. 3 depicts one embodiment of logic associated with updating event data on multiple pages of a computing environment, in accordance with an aspect of the present invention.

Best Mode for Carrying Out the Invention

[0013] In accordance with an aspect of the present invention, an event data update capability is provided in which a portion of a data frame of a browser-displayed page is updated with event data. For example, in a browser with a frameset having a refresh frame (e.g., a hidden, zero-width frame) and a data frame (e.g., a visible frame including an application interface), the refresh frame uses periodic requests to a server to refresh itself to retrieve event data within code understood natively by the browser. A portion of the data frame is updated with the retrieved event data by, for instance, the code executing in the browser.

[0014] As used herein, frame refers to a window within a browser or a separate frame within a window in a browser. That is, a frame is a unit within the browser that can be downloaded from a web server.

[0015] Further, as used herein, a page (e.g., web page) is the entire display generated by a browser window and includes one or more frames. "Code understood natively by a browser" refers herein to code which can be interpreted via one or more language support facilities (e.g., interpreters) built into the browser as originally configured. Since these interpreter facilities are built into the browser, a separate installation or download of these facilities to the browser is not required by the present invention. A refresh frame refers to a frame of a browser-displayed page that is associated with one or more data frames and is used to periodically retrieve event data associated with a portion of a data frame.

[0016] Event data, as used herein, is updated information (e.g., monitoring information) associated with a portion of a data frame displayed by a browser. For example, the portion of the data frame is to be updated with the event data with which it is associated. The event data updating capability descried herein provides, for example, the appearance that updated information is "pushed" from a server to the browser without requiring a manual refresh request by a user. Further, the portion of the data frame to be updated with event data is at least one element of a data frame selectively chosen based on the event data, and is not limited to, for example, the rectangular boundaries of a frame.

[0017] One embodiment of a computing environment incorporating and using one or more aspects of the present invention is described with reference to FIG. 1. As shown in FIG. 1, a computing environment 100 includes one or more computing units 102 coupled to computing units 104 via a connection 106. In one example, connection 106 includes a network connection. One communications protocol used by this connection is HTTP.

[0018] As examples, computing unit 102 includes a desktop or notebook computer, such as a ThinkPad computer offered by International Business Machines Corporation, Armonk, New York. Computing unit 104 includes, for example, an RS/6000 computer system running an AIX operating system offered by International Business Machines Corporation, Armonk, New York, or a UNIX workstation running a UNIX operating system, such as Solaris.

[0019] Computing unit 102 includes a browser 108 and computing unit 104 includes a web server 110 and an associated application 112. As an example, a user of computing unit 102 uses browser 108 to access and display information from application 112. In another embodiment, instead of residing in computing unit 104, application 112 could reside in another computing unit (not shown) which is coupled to computing unit 104. In a computing environment 100 having a plurality of computing units 102, and/or a plurality of computing units 104, the computing units may be homogeneous or heterogeneous to one another. Browser 108 includes, for example, Netscape Navigator, Microsoft Internet Explorer or Mozilla. Web server 110 includes, for instance, WebSphere offered by International Business Machines Corporation.

[0020] In accordance with an aspect of the present invention, one embodiment of logic to be employed in updating event data on a single page is described below with reference to FIG. 2, in which a computing unit 104 includes a web server 202 coupled to application scripts 204, and a computing unit 102 includes a refresh frame 206 coupled to a data frame 208. Refresh frame 206 and data frame 208 are also included in browser 108 (FIG 1; not shown in FIG. 2). As an example, application scripts 204 include Common Gateway Interface (CGI) application scripts.

[0021] When a user utilizes browser 108 to visit a page having information generated by application scripts 204, web server 202 initially loads 210 refresh frame 206 and data frame 208. Data frame 208 is, for example, visible to the user and includes an application user interface. Refresh frame 206 is, for instance, normally hidden, using a frameset setting such as:

```
<FRAMESET cols = "0,* " frameborder= "NO" border = "0">
  <FRAME src = "refresh.cgi" name = "refresh" noresize scrolling = "NO">
  <FRAME src = "data.cgi" name = "data">
</FRAMESET>
```

[0022] Refresh frame 206 can be made visible at certain times (e.g., during a debugging process) by changing the "0" in the above-described frameset setting to a variable that is set to non-zero. Refresh frame 206 and data frame 208 are named in the

above-described frameset setting to provide the frames with the capability to refer to each other. As an alternative to the frameset setting shown above, window objects can be used for defining data frame 208 and refresh frame 206.

[0023] Data frame 208 displays information from application 112 through, for instance, a user interface. This application information may need to be updated. Refresh frame 206 periodically retrieves event data that indicates, for example, application information to be updated on a data frame of a page.

[0024] Refresh frame 206 is set to automatically re-load (i.e., refresh) itself periodically, by including, for example, the following line in the <HEAD> section of its Hypertext Markup Language (HTML) code:

<META HTTP-EQUIV = "REFRESH" CONTENT = "$refreshInterval">

This periodic refresh is performed by periodically sending a refresh request 212 from refresh frame 206 to web server 202. This refresh is automatic because it requires no user intervention, such as a user activating a refresh button on a browser-displayed page.

[0025] An alternate way to specify the refresh of refresh frame 206 is to use a JavaScript function (e.g., window.setTimeout()) to set the window.location property.

[0026] The time interval between the periodic refresh requests 212 is a variable that can be set by a user. This interval can be, for example, fairly short (e.g., 15 seconds) because very little data is transferred in each interval.

[0027] Refresh request 212 causes web server 202 to contact application scripts 204 to determine if new event data exists (e.g., event data that indicates application information in need of updating since the most recent refresh of the refresh frame). If new event data exists, application scripts 204 generate code (e.g., JavaScript) understood natively by browser 108 to update data frame 208. The code includes the new event data and is returned 216 to web server 202. Web server 202, in turn, sends 218 the code, including the new event data to refresh frame 206. Browser 108 executes the code to

update 220 application information according to the new event data (e.g., to update event data displayed in data frame 208). The detection and retrieval of new event data are performed automatically and irrespective of a manual request by a user for event data retrieval or for data frame updating.

[0028] The check for new event data 214 and the generation of the code (e.g., JavaScript code) is performed on computing unit 104 using any server-side language, for example, CGI scripts, Java Server Pages (JSP), Active Server Page (ASP), or PHP Hypertext Preprocessor. This check for new event data and the associated update of the data frame is performed without an application (e.g., server-side application) maintaining state related to one or more browsers or clients. As one example, the following Perl CGI code is generated at application scripts 204 to return new event data to web server 202, and generates JavaScript code which is understood natively by, and is to be executed at, browser 108.

```
my $eventStates = getEvents();

if (scalar(keys %$eventStates))
{
print<< 'END_OF_HTML';
<SCRIPT language = "JavaScript">
<!--
var good = "../images/green-ball-m.gif";
var bad = "../images/red-ball-m.gif";

if (top && top.data && top.data.document && top.data.loadingDone)
{
var doc = top.data.document;
END_OF_HTML

foreach my $k (keys %$eventStates)
    {
    my $i = $$eventStates{$k} ? 'bad' : 'good';
    print "if (doc.$k) {if (doc.$k.src! = $i) {doc.$k.src=$i}}\n";
    }

print << 'END_OF_HTML2';
}
```

```
else { document.write('<p>Data frame not fully loaded yet.  Waiting...</p>');}
//-->
</SCRIPT>
END_OF_HTML2

} # if (scalar(keys %$eventStates))

print "</BODY></HTML>\n";
```

[0029] In the preceding example, new event data, if there is any, is placed in the eventState hashtable. The foreach loop then generates JavaScript to modify red and green images displayed in data frame 208 accordingly. An example of this generated JavaScript code includes the following:

```
<SCRIPT language="JavaScript">
<!--
var good = "../images/green-ball-m.gif";
var bad = "../images/red-ball-m.gif";

if (top && top.data && top.data.document && top.data.loadingDone)
 {
  var doc = top.data.document;

  if (doc.key1) {if (doc.key1.src!=bad) {doc.key1.src=bad}}
  if (doc.key2) {if (doc.key2.src!=good) {doc.key2.src=good}}
  ...

 }
else { document.write('<p>Data frame not fully loaded yet.
Waiting...</p>'); }
//-->
</SCRIPT>
```

[0030] Returning to web server's 202 check to determine whether new event data exists 214, if there is no new event data, application scripts 204 return a minimal frame that includes only a few bytes of HTML code to be sent to refresh frame 206. This small amount of code is required to be in browser 108 to automatically request the next periodic refresh of refresh frame 206. Advantageously, the minimal frame facilitates

minimizing network utilization and load on computing unit 104, which includes web server 202. One example of the minimal frame is the following HTML code, in which "refinterval" would be replaced with the current refresh interval setting:

```
<HTML><HEAD>
<META HTTP-EQUIV="REFRESH" CONTENT="refinterval">
</HEAD><BODY></BODY></HTML>
```

[0031] While data frame 208 is being downloaded into browser 108, the refresh frame can be prevented from modifying data frame 208. This prevention of data frame modification is done, for example, by a check for top.data.loadingDone, as shown in the preceding Perl example. This property is set by data frame 208 in, for example, a JavaScript function that is assigned to the window.onload property.

[0032] If data frame 208 includes links to other pages and a user navigates to one of those pages, the entire frameset (e.g., refresh frame 206 and data frame 208) needs to be replaced so refresh frame 206 does not continue to periodically refresh. This frameset replacement can be accomplished by, for example, associating the following JavaScript code with window.onload:

```
for(var i = 0; i<document.links.length; i++)
{
    var I = document.links[i];
    if(I.target = = null || I.target = = '' || I.target = ='_self') {I.target = '_top';}
}
```

[0033] The preceding JavaScript code example is executed in browser 108 and modifies the target property to be the entire browser window for all links that do not already have a target specifically set.

[0034] In another embodiment, a single refresh frame is used to update a plurality of data frames. Each of the plurality of data frames is included in one of a plurality of pages. As one example of this multiple page case, described with reference to FIG. 3, a computing unit 102 includes data frames 208A, 208B, 208C coupled to a refresh frame

206, and a computing unit 104 includes a web server 202 and associated application scripts 204. Web server 202 initially loads 302 refresh frame 206 and data frames 208A-208C. Data frame 208A is, for instance, visible to a user of computing unit 102 and includes an application user interface. Refresh frame 206, as noted above, is normally hidden from the user's view.

[0035] Following the initial frame loading, data frame 208A directs refresh frame 206 to start monitoring for event data by calling a function 304 (e.g., startMonitoring ( ), typically called from window.onload). This start monitoring function call 304 is necessary because the user can navigate to multiple pages within the frameset including refresh frame 206 and data frames 208A-208C, and some of the pages may be static (i.e., pages that do not require updating with event data). The start monitoring call 304 also passes a parameter, such as a keyword, to refresh frame 206 that identifies the data frame for which event data is to be retrieved.

[0036] Refresh frame 206 then periodically sends a request 306 to refresh itself. This request includes the parameter identifying the data frame. The interval between refresh requests 306 is described above relative to refresh request 212 (FIG. 2). Responsive to refresh request 306, web server 202 queries 308 application scripts 204 to determine if new event data exists and passes the parameter with query 308.

[0037] Application scripts 204 then use the parameter to determine if new event data exists for the data frame identified by the parameter. If no new event data exists, a minimal frame is sent 310 from application scripts 204 to web server 202, and then to a browser in computing unit 102, as described above relative to FIG. 2.

[0038] If new event data exists, generic code is generated by application scripts 204 to be sent 310 to web server 202. For example, application scripts 204 generate the following JavaScript code for creating an array of objects that contain pairs of event names and event values:

```
var a = new Array({name:n1, value:v1}, {name:n2, value:v2},...);
```

In one example, this JavaScript code is inserted into the Perl script described above, and n1, v1, n2, v2, etc. are replaced with actual values when the JavaScript code is generated by application scripts 204.

[0039] The array containing the event data is then sent 312 from web server 202 to refresh frame 206. Refresh frame 206, in turn, passes the array to data frame 208A in a function call 314 (e.g., updatedData(a)).

[0040] Within data frame 208A, one or more event data updates from the array are applied 316 in the data frame. At a later time, data frame 208A calls a function 318 to stop monitoring for event data (e.g., stopMonitoring ( ), typically called from window.onunload). The following function is one example of JavaScript code that is executed in the data frame to apply event data updates from the array of event name-value pairs:

```
function updateData(a)
{
 for(var i = 0; i < a.length; i++)
  {
   var good = "../images/green-ball-m.gif";
   var bad = "../images/red-ball-m.gif";
   var image = a[i].value ? bad : good;
   if (document.images[a[i].key].src!=image)
{document.images[a[i].key].src=image}
  }
}
```

[0041] Advantageously, the event data update technique described above allows application information displayed in a portion of a page to be dynamically and automatically updated while minimizing network utilization and server load, without requiring an entire page or an entire visible frame to be redrawn by the browser, and without requiring user intervention. Further, the present invention provides this update capability without requiring the installation of complex language support software which is not already built into (i.e., native to) the browser. Still further, event data is retrieved

from the server using the same network connection employed to retrieve the refresh frame, thereby eliminating the need for an extra network connection to retrieve event data. In contrast to conventional techniques whereby an entire page is redrawn periodically, the technique described herein avoids erasing or otherwise affecting existing input in user input fields of a page while event data is being applied on the page. Moreover, the updating technique described herein provides a visually appealing presentation of the data frame by not requiring the whole page to temporarily disappear and to be redrawn.

[0042] The capabilities of one or more aspects of the present invention can be implemented in software, firmware, hardware or some combination thereof.

[0043] One or more aspects of the present invention can be included in an article of manufacture (e.g., one or more computer program products) having, for instance, computer usable media. The media has therein, for instance, computer readable program code means or logic (e.g., instructions, code, commands, etc.) to provide and facilitate the capabilities of the present invention. The article of manufacture can be included as a part of a computer system or sold separately.

[0044] Additionally, at least one program storage device readable by a machine embodying at least one program of instructions executable by the machine to perform the capabilities of the present invention can be provided.

[0045] The flow diagrams depicted herein are just examples. There may be many variations to these diagrams or the steps (or operations) described therein without departing from the spirit of the invention. For instance, the steps may be performed in a differing order, or steps may be added, deleted or modified. All of these variations are considered a part of the claimed invention.

[0046] Although preferred embodiments have been depicted and described in detail herein, it will be apparent to those skilled in the relevant art that various modifications, additions, substitutions and the like can be made without departing from the spirit of the

invention and these are therefore considered to be within the scope of the invention as defined in the following claims.